Event Data Models

An Introduction and Survey

Jim Kowalkowski
Marc Paterno

Introduction

What is an Event Data Model?

Why is one useful?

What are common features?

Classes and Instances

- Instance
  - a unit that combines a specific state (data) and the functions used to manipulate it (methods)
- Class
  - a type that defines related instances
    - a description of what the instances have in common (types of data, method definitions)
  - the body of code that manipulates the data in the instances
- A program can have multiple instances of the same class, each with different values

Parameterized Classes

- Class template
  - A description for how to write a class
  - Describes a family of classes that share common characteristics
  - Instantiating a class template causes the compiler to write a class; one can then make instances of the class
    - *std::vector* — class template
    - *std::vector<float>* — instantiated class
    - *std::vector<float> vf* — object, or instance

What is an Event Data Model?

- An Event Data Model (EDM) provides a mechanism for managing data related to an physics event within a program
- An EDM is *not*:
  - a persistency mechanism
  - an I/O mechanism
  - a file format

… although it is related to all of these things

Why is an EDM Useful?

- It allows for independence of reconstruction modules
  - This assumes a modular framework
  - Modules communicate only via the EDM
    - true whether modules are C++ or Fortran
  - Modules can be developed and maintained independently – critical for maintainability of a large body of code

Why is an EDM Useful?

- Can isolate users from need to interact with persistency mechanism
  - implementation of streaming
- Can isolates users from I/O mechanism
  - details of reading files
- Can isolates users from changes in file formats

General Features

- Some features are shared by all EDMs
  - *Event* class, collection of data for one event
  - Many classes representing various "pieces" of an event, and collections thereof:
    - tracking hits; calorimeter energies
    - tracks, candidate particles (electron, tau, jet, ...)
  - Navigation classes
    - efficient location of specific "pieces"
    - associations between "pieces" of the Event
  - Metadata classes

Common Needs

- More than one algorithm can produce each kind of output
  - need to be able to hold, and uniquely identify, the output of a specific algorithm
    - *e.g.* cone algorithm jets and KT algorithm jets
  - A single algorithm can be configured with different parameters; need to distinguish
    - *e.g.* R=0.7 cone jets and R=0.4 cone jets

Common Needs

- Many different types of reconstructed "pieces" need to be stored in the event
- All these types make up "the EDM"
- Continuous need to add new types of "pieces" to the event
  - it is impossible to predict them all at the outset of the experiment
  - the EDM grows as the need arises
- Sometime we call the *core* classes "the EDM"

Identifying BTeV Requirements

- "You can get at the data, whatever language you speak"
  - in the trigger? offline?
- "Data structures should have fixed maximum sizes"
  - goal is speed – time not wasted allocating and freeing memory
  - can be achieved in different manners, allowing one to retain a flexible EDM
- Full data access for Fortran, no copying

Mission Impossible?

1. Trigger code must access data without requiring any copying of data
2. It must be possible to write triggers in Fortran 77

- Why not both?
  - Fortran common blocks are disconnected from an object-based EDM
  - Tremendous difficulty mapping even simple C++ structures into Fortran

Before Designing an EDM

- Need to start with requirements
  - required features
  - attractive features
  - priorities
- Possible to modify an existing EDM, or design from scratch
- An overview of some existing data models may help illustrate the range of possibilities …

The Survey

A tour through the major features of the CDF, DØ, Gaudi and MiniBooNE event models

- A more detailed document on this topic shall be available, at:

  http://www-cdserver.fnal.gov/public/cpd/aps/EDMSurvey.htm

- This survey is an extract of the tables from the current version of that document
- Please contact the authors with any corrections
  - paterno@fnal.gov & jbk@fnal.gov

Overview

- The CDF and DØ EDMs are in active use by those experiments, respectively
- The Gaudi EDM is under development by the LHCb experiment
- The MiniBooNE EDM is in active use, but still undergoing development. MiniBooNE uses both C++ and Fortran
  - Features viewed from C++: MB
  - Features viewed from Fortran: MBF

Access to the Event

How does a user gain access to an Event?

- CDF passed into functions; also global
- DØ passed into functions
- Gaudi search in global registry
- MB passed into functions
- MBF globally available

- Global access will have some influence on ability to handle *multiple events*

Event Multiplicity

During development, testing, and simulation, it is sometimes useful to handle more than one Event at a time

Can we have more than one Event?

- CDF Yes, but use of global causes trouble
- DØ Yes
- Gaudi Not yet; plans are to access "named" instances
- MB Yes
- MBF No; too hard to do in Fortran

Definition of Event Data Object

- The *Event* is a container of objects
  - raw data; MC particles; GEANT hits
  - trigger results, reconstructed objects
- Each experiment has its own terminology for the constituents of an *Event*
  - CDF storable objects
  - DØ chunks
  - Gaudi data objects
  - MB chunks
- Often, the things the *Events* collects are themselves collections (of hits, tracks, jets ...)

Event Interface

What is the "look and feel" of an Event?

- CDF collection with "generic" iterator
- DØ "database" with type safe queries
- Gaudi filesystem-like hierarchy of named nodes
- MB associative array of type safe nodes
- MBF subroutine calls to load common blocks

Adding to the Event

How is a new object added to an Event?

- CDF ownership passed (design), no copy
- DØ ownership passed (design), no copy
- Gaudi ownership passed (convention), no copy
- MB ownership passed (design), no copy
- MBF copy from common block to C++ object, then as above

- Relying on convention is error prone!

Mutability of Event Data

Can objects in the Event be modified?

- Desire for reproducibility argues this should be very tightly controlled
  - CDF no, except that collections can grow
  - DØ no
  - Gaudi yes
  - MB *under development*
  - MBF *under development*

Inheritance

Is inheritance from a base class needed?

- CDF   from *TObject* via *StorableObject*
  - must implement a streamer; requires CDF macro, to write some of the interface required by ROOT
- DØ   from *do_Object* via *AbsChunk*
  - requires DØ macro, to write some of the interface required by DOOM; requires possession of various IDs

Inheritance (cont'd)

- Gaudi from *DataObject*
  - must be able to return a globally unique ID for the class.
- MB none
  - Should be a POD; current usage of ROOT violates this
- MBF none
  - Any properly padded common block, no strings allowed

EDO Multiplicity

Is it possible to access more than one instance of an EDO class at one time?

- Everyone needs this
  - CDF tracks: needs more than one set, several competing algorithms
  - DØ raw data: need more than one in simulation
- This ability generates a requirement for labelling EDOs.

EDO Multiplicity (continued)

Is it possible to access more than one instance of an EDO class at one time?

• CDF	yes
• DØ	yes
• Gaudi	yes
• MB	yes
• MBF	no

Labelling

How are objects in an Event labelled?

- CDF
  - Unique object ID, configuration parameter set ID, descriptive string, class version, and class name
- DØ
  - Unique object ID, configuration parameter set ID, parent object IDs, geometry & calibration IDs, and string labels

Labelling (cont'd)

- Gaudi
  - Class ID, descriptive string with hierarchical path
- MB
  - Descriptive string and class name
- MBF
  - Descriptive string

Query Interface

How does a user specify which EDO he wants?

- CDF
  - Custom iterators with optional selectors specifying a combination of labels
- DØ
  - User specified criteria based on object data or specific labelling information; multiple objects returned

Query Interface (cont'd)

- Gaudi
  - string path information
- MB
  - Class name/descriptive string; single object returned
- MBF
  - Descriptive string; single object put into common block

Query Results

In what form is the result returned?

- CDF
  - Custom iterator; read-only access to the object they refer to and traversal to next object
- DØ
  - Collection of handles that allow read-only access to the objects

Query Results (cont'd)

- Gaudi
  - Bare pointer to the base class object or to the object itself
- MB
  - Read-only pointer to the object
- MBF
  - Populated common block, a copy of the event data

Multiple Matches

What happens if more than one EDO matches the query?

- CDF iterator moves through the matches
- DØ collection of matches is returned
- Gaudi *not applicable*
- MB no multiple matches implemented
- MBF no multiple matches allowed

Support for Associations

What support is given for making associations between EDOs?

- Bare pointers are unsuitable
  - When a pointed-to object is deleted
  - When only parts of an *Event* are written
  - When reading an *Event*
- "Smart pointers" of various sorts are the usual solution
  - class templates with special behavior

Parameterized Classes

- Class template
  - A description for how to write a class
  - Describes a family of classes that share common characteristics
  - Instantiating a class template causes the compiler to write a class; one can then make instances of the class
    - *std::vector* — class template
    - *std::vector<float>* — instantiated class
    - *std::vector<float> vf* — object, or instance

Support for Associations

- CDF
  - Special link classes that are converted from pointer to id and back automatically; links exist for objects with collection associations
- DØ
  - Special link classes that are converted from pointer to id and back semi-automatically; link classes exist for top-level EDOs and for items within collections

Support for Associations (cont'd)

- Gaudi
  - Special link classes that re converted from pointer to id automatically; links exists for *DataObjects* or vectors
- MB
  - currently no infrastructure support

Restrictions on Associations

- In all cases, C++ object models disallow (by convention) use of bare pointers
- Associations are one-way, from “newer” objects to “older” objects
  - enforced for CDF, DØ; convention for Gaudi
- Complex associations must be implemented in distinct EDOs

Persistency Impositions

What requirements are placed on EDOs by the persistency mechanism?

- CDF macros, streamers, *TObject*
- DØ macros, *do_Object*
- Gaudi all data public, or available with get/set methods
- MB macros
- MBF C struct, padded to map to common block

I/O Format

What file format is used?

- CDF ROOT
- DØ DSPACK is standard, others are possible
- Gaudi Objectivity and ROOT
- MB ROOT
- MBF ROOT

- Multiple I/O formats are available for those designs that have isolated the persistency mechanism from the EDM

Schema Evolution

- Mentioned several times as important
  - New classes are added – easy!
  - Existing classes are changed – harder
- Widely different degrees of automation
  - CDF *if* statements in streamers
  - DØ automated, using DoOM data dictionary
  - Gaudi *if* statements in converters
  - MB automated, using ROOT data dictionary

Translation Mechanism

What is done to write out/read in an object?

- CDF
  - Hand written code to write object's data into the ROOT buffer; transient representation typically differs significantly from the persistent form
- DØ
  - Automated by data dictionary; copies data to the Fortran bank structure, then to output. Rarely used activate/deactivate can do simple transient mapping.

Translation Mechanism (cont'd)

- Gaudi
  - Converter external to the class reads state out into the persistency package buffers; copy the data objects into objectivity objects, then write the those objects
- MB
  - Automated by data dictionary, copies data to ROOT buffers.

Where to go from here?

Questions for BTeV

- Are your requirements agreed upon?
  - If not how will consensus be reached
  - If so, are they clearly expressed?
- What process will be used to move from requirements to a solution?
  - Concrete milestones
  - Time estimates
  - Continuous review of both to keep project on track